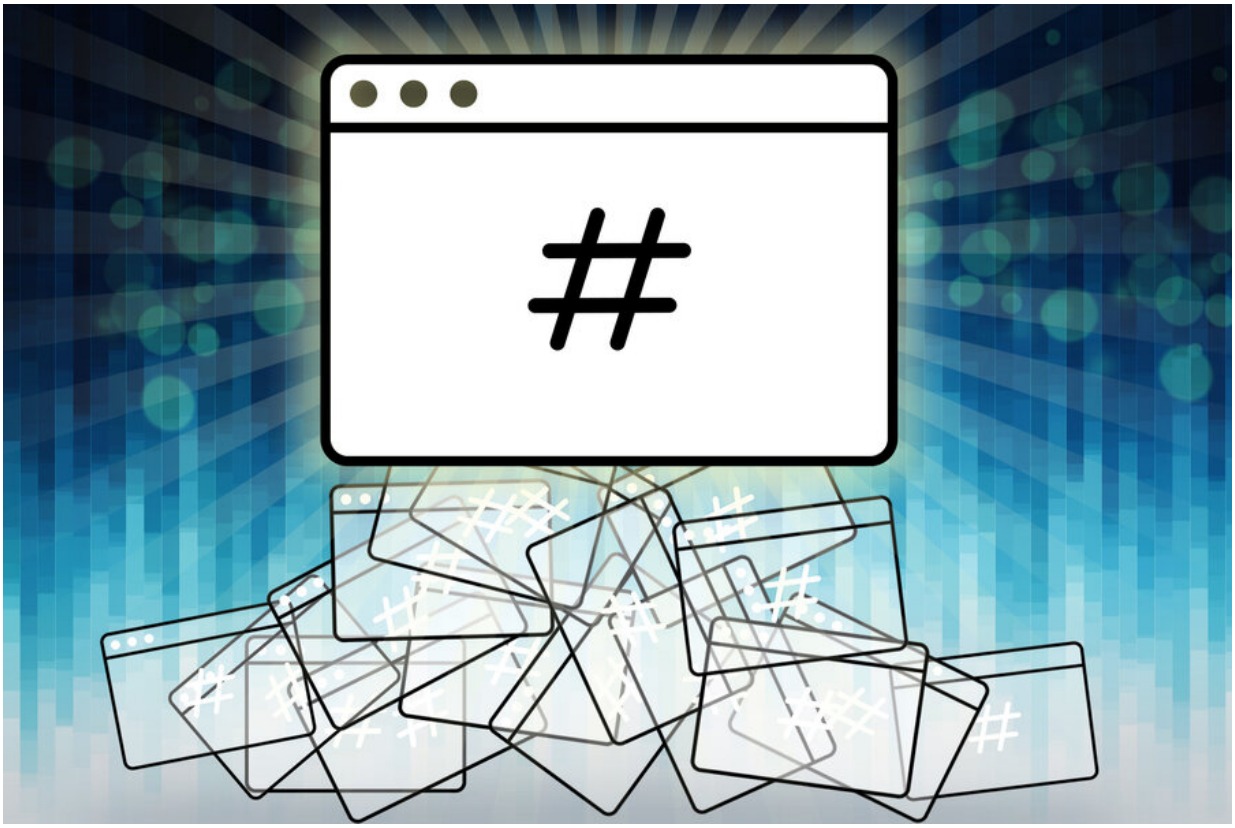
A new method to boost the speed of online databases

March 13 2023, by Adam Zewe



Researchers from MIT and elsewhere set out to see if they could use machine learning to build better hash functions. Credit: Christine Daniloff, MIT

Hashing is a core operation in most online databases, like a library catalog or an e-commerce website. A hash function generates codes that

replace data inputs. Since these codes are shorter than the actual data, and usually a fixed length, this makes it easier to find and retrieve the original information.

However, because traditional hash functions generate codes randomly, sometimes two pieces of data can be hashed with the same value. This causes collisions—when searching for one item points a user to many pieces of data with the same hash value. It takes much longer to find the right one, resulting in slower searches and reduced performance.

Certain types of hash functions, known as perfect hash functions, are designed to sort data in a way that prevents collisions. But they must be specially constructed for each dataset and take more time to compute than traditional hash functions.

Since hashing is used in so many applications, from database indexing to data compression to cryptography, fast and efficient hash functions are critical. So, researchers from MIT and elsewhere set out to see if they could use machine learning to build better hash functions.

They found that, in certain situations, using learned models instead of traditional hash functions could result in half as many collisions. Learned models are those that have been created by running a machine-learning algorithm on a dataset. Their experiments also showed that learned models were often more computationally efficient than perfect hash functions.

"What we found in this work is that in some situations we can come up with a better tradeoff between the computation of the hash function and the collisions we will face. We can increase the computational time for the hash function a bit, but at the same time we can reduce collisions very significantly in certain situations," says Ibrahim Sabek, a postdoc in the MIT Data Systems Group of the Computer Science and Artificial

Intelligence Laboratory (CSAIL).

Their research, which will be presented at the International Conference on Very Large Databases, demonstrates how a hash function can be designed to significantly speed up searches in a huge database. For instance, their technique could accelerate computational systems that scientists use to store and analyze DNA, amino acid sequences, or other biological information.

Sabek is co-lead author of the paper with electrical engineering and computer science (EECS) graduate student Kapil Vaidya. They are joined by co-authors Dominick Horn, a graduate student at the Technical University of Munich; Andreas Kipf, an MIT postdoc; Michael Mitzenmacher, professor of computer science at the Harvard John A. Paulson School of Engineering and Applied Sciences; and senior author Tim Kraska, associate professor of EECS at MIT and co-director of the Data Systems and AI Lab.

Hashing it out

Given a data input, or key, a traditional hash function generates a random number, or code, that corresponds to the slot where that key will be stored. To use a simple example, if there are 10 keys to be put into 10 slots, the function would generate a random integer between 1 and 10 for each input. It is highly probable that two keys will end up in the same slot, causing collisions.

Perfect hash functions provide a collision-free alternative. Researchers give the function some extra knowledge, such as the number of slots the data are to be placed into. Then it can perform additional computations to figure out where to put each key to avoid collisions. However, these added computations make the function harder to create and less efficient.

"We were wondering, if we know more about the data—that it will come from a particular distribution—can we use learned models to build a hash function that can actually reduce collisions?" Vaidya says.

A data distribution shows all possible values in a dataset, and how often each value occurs. The distribution can be used to calculate the probability that a particular value is in a data sample.

The researchers took a small sample from a dataset and used machine learning to approximate the shape of the data's distribution, or how the data are spread out. The learned model then uses the approximation to predict the location of a key in the dataset.

They found that learned models were easier to build and faster to run than perfect hash functions and that they led to fewer collisions than traditional hash functions if data are distributed in a predictable way. But if the data are not predictably distributed, because gaps between data points vary too widely, using learned models might cause more collisions.

"We may have a huge number of data inputs, and each one has a different gap between it and the next one, so learning that is quite difficult," Sabek explains.

Fewer collisions, faster results

When data were predictably distributed, learned models could reduce the ratio of colliding keys in a dataset from 30% to 15%, compared with traditional hash functions. They were also able to achieve better throughput than perfect hash functions. In the best cases, learned models reduced the runtime by nearly 30%.

As they explored the use of learned models for hashing, the researchers

also found that throughout was impacted most by the number of sub-models. Each learned model is composed of smaller linear models that approximate the data distribution. With more sub-models, the learned model produces a more accurate approximation, but it takes more time.

"At a certain threshold of sub-models, you get enough information to build the approximation that you need for the hash function. But after that, it won't lead to more improvement in collision reduction," Sabek says.

Building off this analysis, the researchers want to use learned models to design hash functions for other types of data. They also plan to explore learned hashing for databases in which data can be inserted or deleted. When data are updated in this way, the model needs to change accordingly, but changing the model while maintaining accuracy is a difficult problem.

"We want to encourage the community to use machine learning inside more fundamental data structures and operations. Any kind of core data structure presents us with an opportunity use machine learning to capture data properties and get better performance. There is still a lot we can explore," Sabek says.

More information: Can Learned Models Replace Hash Functions? www.vldb.org/pvldb/vol16/p532-sabek.pdf

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology